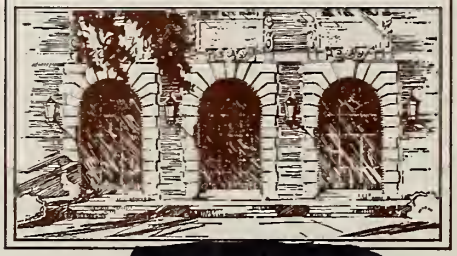
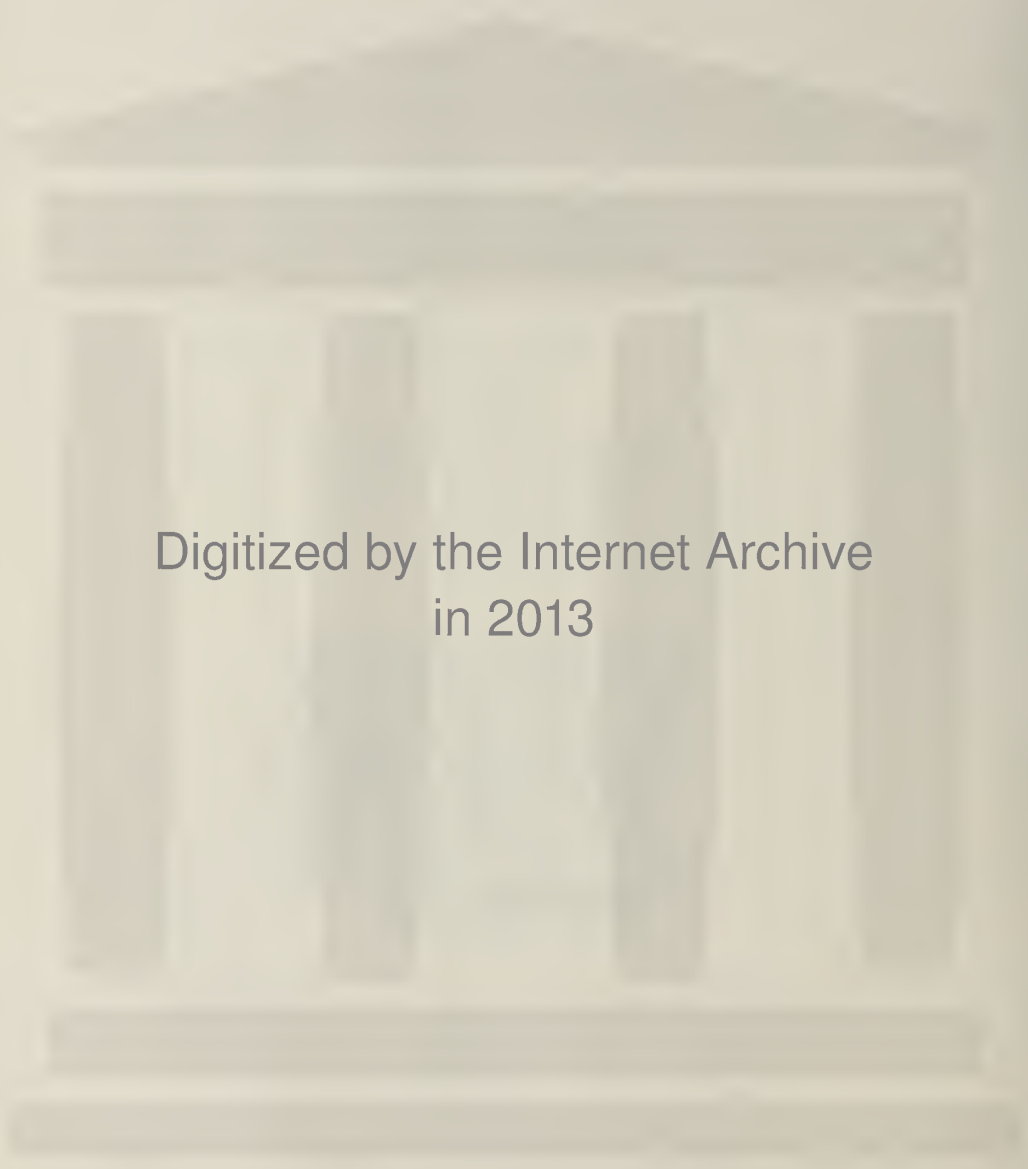




Report No. UIUCDCS-R-75-703 NSF - OCA - GJ-328 - 000003

THE EIGENVALUE PROBLEM IN THE OL/2 LANGUAGE

by

Ricardo Macias Carrasco

May 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-75-703

THE EIGENVALUE PROBLEM IN THE OL/2 LANGUAGE*

by

Ricardo Macias Carrasco

May 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF-GJ-328 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, May 1975.

ACKNOWLEDGMENT

I wish to thank Professor J. R. Phillips, creator of OL/2, for suggesting this topic and for his helpful suggestions and guidance. I would also like to thank D. R. Jurich, member of the OL/2 team who contributed to the effort, and the University of Illinois at Urbana-Champaign for the support it has provided.

TABLE OF CONTENTS

1. INTRODUCTION

OL/2 (Operator Language/Version 2) [4] has been specifically designed to write array algorithms in a natural and efficient way. It adheres to the common mathematical language and is therefore ideal for solving problems in linear algebra.

This thesis is concerned with the implementation in the OL/2 language of the well-known eigenvalue problem. The eigenvalue problem consists of finding the nontrivial solutions of

$$Av = \lambda v \tag{1}$$

where A is a matrix of order n, v is a vector of order n, and λ is a scalar.

The problem, as stated, has a number of ramifications which require careful consideration. We will treat the problem with the objective of providing the framework for a very high level language implementation in the context of the OL/2 language.

1.1 General Features

The general eigenvalue problem gives rise to a number of "sub-problems". For example, in (1) a user may require the computation of all eigenvectors and eigenvalues $\{v_i, \lambda_i\}$ $1 \leq i \leq n$, only the eigenvalues $\{\lambda_i\}$ $1 \leq i \leq n$, or only a particular subset of these. Therefore, in this implementation we seek to provide a set of OL/2 statements that allows the various possibilities to be computed by specifying only what has to be computed without specifying how it is computed. The question of how to compute, that is, how to choose the algorithms, is determined entirely within the eigensystem module which is part of the OL/2 implementation.

In these statements we have taken advantage of the facilities that the OL/2 language provides to write array expressions and to reference arrays. For example, if the user wants to compute the set of eigenvalues $\{\lambda_i\}$ $1 \leq i \leq n$, of a given matrix A, he would write the following OL/2 statement

```
EIGENVALUES A → V;
```

where V is a one-dimensional array that will contain the eigenvalues of A. Similar statements are provided for other possible cases.

As far as the implementation of the general eigenvalue problem (1) is concerned, we will restrict ourselves to a real matrix and complex V and λ. In sections 2, 3 and 4 we will discuss how the problem is solved in general and how it relates to the current implementation in OL/2.

1.2 Earlier Work

Some work has been done in this area but with a traditional approach. The most important is the subroutine package EISPACK [8] that uses a more primitive approach. The difference is that the user has to know for example the details that involve each particular problem. For example, if he wishes to get the solutions for (1) he must provide additional JCL statements, and be able to write a call statement involving the exact description of the type of computation he is requesting together with the different operands involved.

For example, to get the solution $\{\lambda_i\}$ $1 \leq i \leq n$ of (1) in a FORTRAN program, he has to include the statement

```
CALL EISPACK(NM, N, MATRIX ('REAL', AR), VALUES (WR, WI))
```

where NM and N specify dimensions for the previously defined arrays AR, WR and WI.

In OL/2 we free the users of these details and provide greater flexibility in solving this problem by including a number of features that will be discussed in the next sections.

2. ARRAY STRUCTURES

2.1 Data Types

Before we consider the eigenvalue problem, it is necessary to give an explanation of the different data types which are available in OL/2, together with the information structures that control the use and access of these data types.

The language provides for the declaration of scalars, various types of arrays, vector spaces, and sequences of arrays. In this thesis we denote arrays, usually matrices, by capital letters A, B, C, vectors by small letters x, y, z, and scalars by Greek letters α β γ.

Consider a matrix A with elements $\alpha_{i,j}$ $1 \leq i, j \leq n$. Table I lists the different geometric shapes that are defined in OL/2. By definition, if the geometric shape is not specified in a declaration, then it is assumed to be a square matrix.

Table I. Basic Data Types for Square 2-Dimensional Arrays

Geometric Shape of A	Abbreviation	Stored Elements
Strictly Lower Triangular	SLT	$1 \leq j < i \leq n$
Lower Triangular	LT	$1 \leq j \leq i \leq n$
Diagonal	D	$1 \leq i = j \leq n$
Strictly Upper Triangular	SUT	$1 \leq i < j \leq n$
Upper Triangular	UT	$1 \leq i \leq j \leq n$
Tridiagonal	T	$\lvert i-j \rvert < 1 \quad 1 \leq i,j \leq n$
Square	S	$1 \leq i \leq n \quad 1 \leq j \leq n$
Rectangular	R	$1 \leq i \leq m \quad 1 \leq j \leq n$

Some typical declarations are the following:

```
LET T BE A TRIDIAGONAL MATRIX OF ORDER (N);
LET X AND Y BE VECTORS OF ORDER (M);
LET S BE A SCALAR;
LET L I_K_I BE A SEQUENCE MOD (K) OF LOWER
     TRIANGULAR MATRICES OF ORDER (N).
```

For further details of the OL/2 language and its use, see [4, 6].

2.2. Array Control Blocks

In this section we describe the internal control structure associated with the various types of arrays [1, 7].

All arrays in OL/2 are described at execution time by an array control block (ACB) which is used to pass information to the various computational routines and to control the partition data structure. For every array explicitly delcared, an ACB is allocated and all the references to this array are through this block. Figure 1 shows the structure of the ACB and the various fields are described as follows:

#NAME	The name of the array as actually defined. This field is present in the ACB for diagnostic purposes.
#ATTRIBUTES	The arithmetic attributes of the data stored in the array.
#DIM	The number of dimensions of the array.
#MOD	Used to specify sequences of arrays.
#TYPE	The geometric shape as described in section 2.1.

#ROW-INCR #DIAG-INCR	Constants related to #TYPE which are used by the computational routines.
#LENGTH	The number of elements of the array.
$ORIGIN	A pointer to the actual storage location of the first element of the array.
#PARTITION-PTR	A pointer to the Partition Control Block (PCB) described in [1, 7].
#EXTENT #LOWER #UPPER	The extent, lower bound, and upper bound respectively for one array. There is one triple for each dimension up to the number of dimensions specified in #DIM.

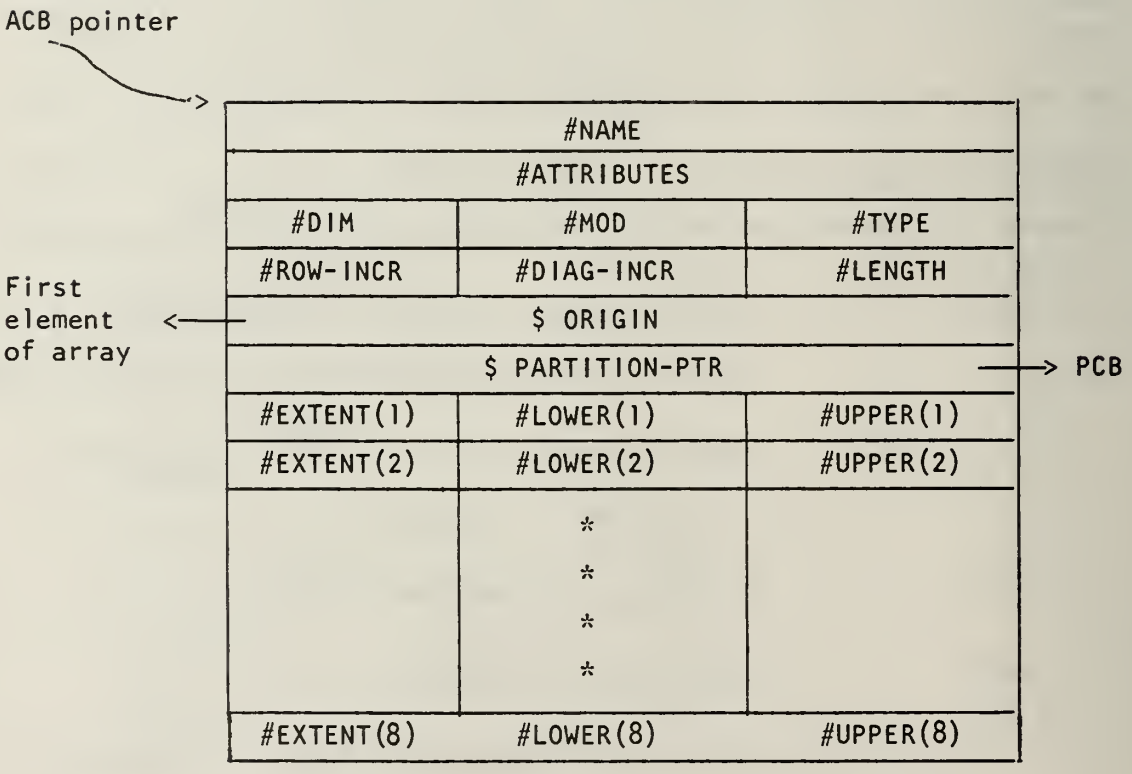


Figure 1. Array Control Block

It has to be pointed out that all data types are stored as linear arrays and are accessed at any time by means of $ORIGIN, #DIAG-INCR and #ROW-INCR variables in the ACB. For further explanation of how this is accomplished, refer to [5, 7].

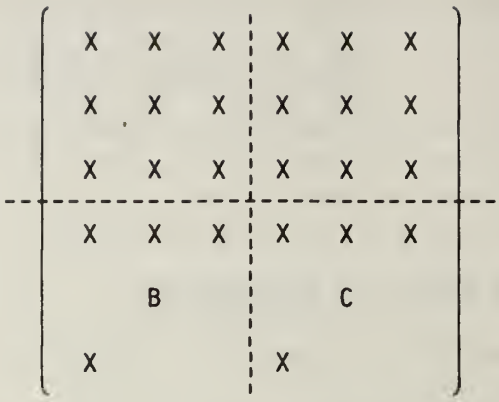
2.3 Partitioning and Array Expressions

Together with the concept of geometric shape for an array is the concept of partitioning an array. This is implemented in the language and is used to separate an array into subarrays by defining partition lines between specified rows and columns of the array. This allows a user to reference subarrays as independent arrays.

The OL/2 language provides for a very general and dynamic partitioning of any type of array. This capability of the language among other things allows us to rename arrays without actually allocating storage for them. To illustrate this, we consider a typical example of a partition statement and associated LET and SET statement.

```
LET A BE A MATRIX OF ORDER (7);
PARTITION A AFTER ROW 3 AND COLUMN 2;
SET B = A < 2,1 >,   C = A < 2,2 >;
```

It should be clear from these statements that the usual idea of partitioning is easily described by a user. The convention of using A < 2,1 > for the subarray A_{21} is simply a matter of convenience. The partitioned array is shown below.



It is important to realize that we can rename part of an array and treat it independently and not allocate storage for it. In [7] partitioning is explained in detail and includes the dynamic aspects where the position lines may be moved over the parent array A in a nearly arbitrary manner.

The different data types and the arrays resulting from partitioning, together with the arithmetic operators $\pm$, *, /, ' (transpose) and PL/1 functions, are used to form OL/2 arithmetic array expressions. Table II outlines the types and operators that may be used to construct expressions. At this point it is necessary to remark that OL/2 treats column and row vectors as degenerate arrays. In [3] the compilation of OL/2 arithmetic expressions is explained in detail. Our main concern as far as these expressions are concerned is the possible legal expressions that can be formulated, and these are of course any combination of data types and operators in infix form that do not violate the conventional rules of linear algebra, namely multiplication, division and addition between scalars, vectors and matrices.

Table II. OL/2 Operations

OL/2 SYMBOL	MEANING	OPERAND 1	OPERAND 2	RESULT	EXAMPLE
'	vector or array transpose	col vector	_____	row vector	
		row vector	_____	column vector	x'
		array	_____	array	
+ -	unary negation for each element in array	array	_____	array	-C
(,)	vector inner product	col vector	col vector	scalar	(x,y)
*	vector inner product	col vector	col vector	scalar	x'*y
|| ||	various vector	vector	_____	scalar	|| x ||
	and matrix norms	matrix	_____		
* + - ** /	scalar operations	scalar	scalar	scalar	$\alpha + \beta$
*	array times col vector or row	array	col vector	col vector	A*z
	vector times array	row vector	array	row vector	z'*A
	array multiplication	array	array	array	A*B
	vector outer product	col vect	row vect	array	x*y'
	vector times scalar	vector	scalar	vector	
	array times scalar	array	scalar	array	α*A
+ -	array sum	array	array	array	A - B
=	scalar ← scalar	scalar	scalar	scalar	$\alpha = \gamma$
	array ← array	array	array	array	A = C

The following are examples of OL/2 array expressions:

```
A = z * z'          ;
B = α * A * x * y'  ;
x = β * A * z       ;
α = (x,y)           ;
```

With these concepts, we may now proceed to the main topic of this thesis, namely, the eigenvalue problem and how we can control the computation of the eigenvalues and eigenvectors within the context of the OL/2 language.

3. THE EIGENVALUE PROBLEM

3.1 Definition

The general eigenvalue problem can be stated as follows:

Let V be a vector space over a field F. Given an integer n and an n by n matrix A, find those scalars $\lambda_i \in F$ and vectors $v_i \in V$ such that

$$A v_i = \lambda_i v_i \qquad 1 \leq i \leq n \qquad (1)$$

This is one of the most important problems in computational linear algebra, and is of practical as well as theoretical interest.

We recall that the problem as defined has as a solution a set of n scalars λ_i and associated vectors v_i for $1 \leq i \leq n$. In practice, however, users may be interested in computing:

i) all the eigenvalues and eigenvectors of A.

ii) the eigenvalues of A only.

iii) the k smallest (or k largest) eigenvalues of A.

iv) the eigenvalues of A in an interval (α, β).

v) the eigenvalues and eigenvectors of A in an interval (α, β).

The solution to the problem of how to choose the appropriate algorithm for each case of these cases depends largely on the characteristics of the matrix A. We will discuss this in the next section.

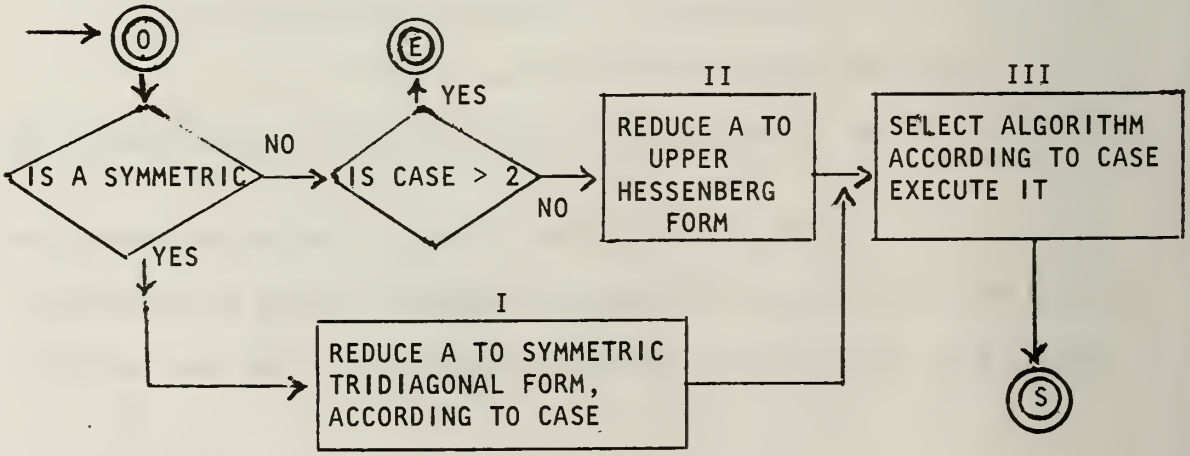
3.2 Algorithm Selection and Control

The solution of the problem of choosing appropriate algorithms is mainly determined by the symmetry and sparseness of the matrix A.

A may be symmetric or unsymmetric. In the former case we will be able to solve the five problems as stated above, while in the latter we will be more restricted. A may also be dense or sparse (band matrix). Since the present implementation of OL/2 does not allow complex, band, or randomly sparse matrices, we will be concerned only with the case in which F is the field of real numbers and A is a dense unsymmetric or symmetric matrix. This does not mean that we cannot store a band or sparse matrix in a square array, but means that the algorithms selected will not be the adequate ones. Tridiagonal matrices are optimally stored.

The five cases labeled (i) through (v) above will be denoted below by the value of CASE, namely, 1, 2, 3, 4 or 5, respectively.

The following flow diagram shows the way in which the problem is organized.



E means in the above diagram that the state-of-the-art does not solve the problem satisfactorily for those cases.

Next we will explain how operations are performed in the boxes labeled I, II and III respectively. All routines which are referenced belong to the EISPAC subroutine package [8].

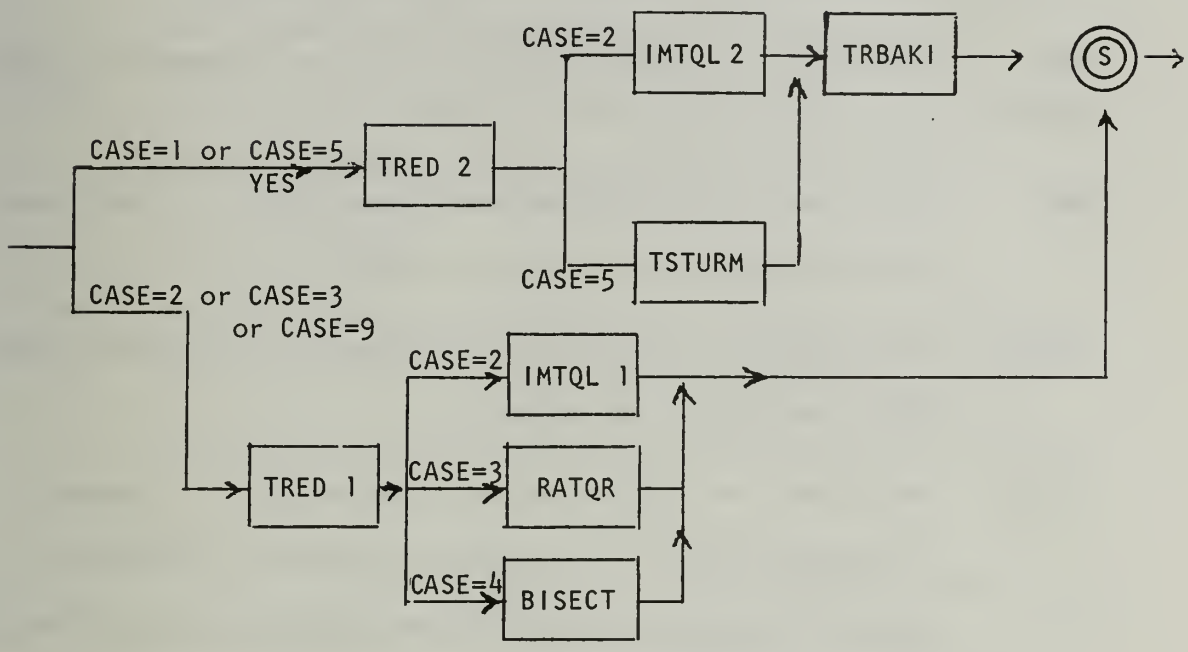


Figure 2. Path I, III

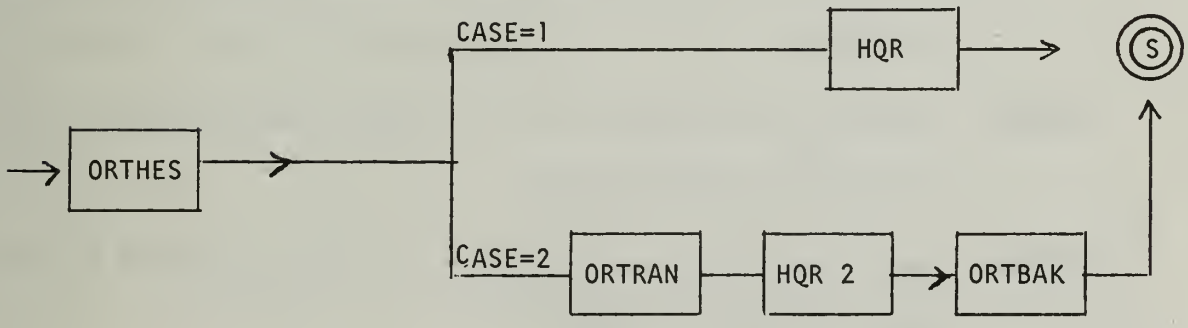


Figure 3. Path II, III

Next, we list the EISPACK routines mentioned in Figs. 2 and 3 and reference their computational characteristics.

BISECT: Determines those eigenvalues of a symmetric tridiagonal matrix in a specified interval using Sturm sequencing.

HQR: Determines the eigenvalues of a real upper Hessenberg matrix using the QR method.

HQR2: Determines the eigenvalues and eigenvectors of a real upper Hessenberg matrix using the QR method.

IMTQL1: Determines the eigenvalues of a symmetric tridiagonal matrix using the implicit QL method.

IMTQL2: Determines the eigenvalues and eigenvectors of a symmetric tridiagonal matrix. It uses the implicit QL method to compute the eigenvalues and accumulates the QL transformations to compute the eigenvectors.

ORTBAK: Backtransforms the eigenvectors of a real general matrix from the eigenvectors of that upper Hessenberg matrix determined by ORTHES.

ORTHES: Reduces a real general matrix to upper Hessenberg form using orthogonal transformations.

ORTRAN: Accumulates the transformations in the reduction of a real general matrix by ORTHES.

RATQR: Determines the algebraically smallest or largest eigenvalues of a symmetric tridiagonal matrix using the rational QR method with Newton corrections.

TRBAK1: Forms the eigenvectors of a real symmetric matrix from the eigenvectors of that symmetric matrix determined by TRED1.

TRED1: Reduces a real symmetric matrix to a symmetric tridiagonal matrix using orthogonal similarity transformations.

TRED2: Reduces a real symmetric matrix to a symmetric tridiagonal matrix using and accumulating orthogonal similarity transformations.

TSTURM: Determines those eigenvalues of a symmetric tridiagonal matrix in a specified interval and their corresponding eigenvectors, using Sturm sequencing and inverse iteration.

The reader may refer to [9] which gives the mathematical details about these routines or to [8] for a more detailed explanation.

So far we have seen the general framework that can be used to solve the eigenvalue problem. We now proceed to discuss the implementation of this problem in OL/2 using the very high level characteristics of the language.

4. THE EIGENVALUE PROBLEM IN OL/2

4.1 Eigenvalue Statement

In this section we will describe the implementation of the eigenvalue problem in the current version of the OL/2 language.

As was mentioned earlier, given an array A, the main interest is in computing:

(i) eigenvalues and eigenvectors

(ii) eigenvalues

(iii) extreme (smallest or largest) eigenvalues

(iv) eigenvalues in an interval

(v) eigenvalues and eigenvectors in an interval

The corresponding OL/2 statements for each one of these cases are described next.

(i') EIGENSYSTEM A → U, V, B;

(ii') EIGENVALUES A → U, V;

(iii') EIGENVALUES A SMALLEST k → U;

EIGENVALUES A LARGEST k → U;

(iv') EIGENVALUES A IN (α, β) → U;

(v') EIGENSYSTEM A IN (α, β) → U, B;

We will see how to interpret one of these OL/2 statements, say (i'), the others being interpreted in an analagous way.

For statement (i), recall that the matrix A can be symmetric or unsymmetric. In the former case we will have real eigenvalues and in the latter a mixture of real and complex eigenvalues. Therefore, for a

matrix of order n, we will expect n eigenvalues and eigenvectors, and we interpret (i') as: compute the eigenvalues and eigenvectors of the matrix A; for the eigenvalues, place the real parts in vector U, imaginary parts in vector V, and place the eigenvectors in the array B. Of course, if we know in advance that A is symmetric, then (i') reduces to

EIGENVALUES A → U, B;

The only restriction here is that U and V have to be declared as vectors of order n and B as a matrix of order n. The same reasoning applies to the other statements.

Because of the different geometric shapes allowed in the OL/2 language, the partitioning capabilities, and the ease of forming array expressions, we have a very general capability within the eigenvalue statements:

a) A may be any OL/2 array expression or part of an array.

b) U,V,B may be OL/2 arrays or part of some array (subarray).

c) α,β,k may be any OL/2 scalar expression.

It can be seen that these facilities allow for a variety of eigenvalue computations, and simultaneously free the user of all details that arise with those computations.

The following is an example of part of an OL/2 program that involves eigenvalue statements:

```
LET A AND B BE MATRICES OF ORDER (7), S BE A SCALAR,
    U AND V BE VECTORS OF ORDER (7), AND W BE A VECTOR
    OF ORDER (2);                    INPUT A, S;
PARTITION A AND B AFTER ROW 1 AND COLUMN 1;
SET L TO THE LOWER TRIANGULAR PART OF A;
```

```
EIGENSYSTEM L → U, B;
                                    OUTPUT U, B;
PARTITION U AND V AFTER ROW 1;
EIGENVALUES A < 2,1 > * A < 1,2 > → U < 2 >, V < 2 >, B < 2,2 >;
                                    OUTPUT U < 2 >, V < 1 > , B < 2,2 > ;
EIGENVALUES L SMALLEST 2*S → W;
                                    OUTPUT W;
```

The rest of this section will be devoted to explain the compilation of the eigenvalue statement.

4.2 TACOS

The compiler for OL/2 is generated by a general translator writing system called TACOS [2]. TACOS is capable of taking as input the syntax and the semantics of a language and constructing a linked list completely specifying them. A fixed interpretative parser which performs a top-down parse according to the contents of the list makes the compiler complete.

The syntax is specified in IBNF (Informal Backus Normal Form) similar to BNF. The differences between IBNF and BNF are the following:

(i) Use of parenthetical expressions which considerably reduce the number of phrase classes.

(ii) Three different repetition characters are defined to allow the phrase class interpretations given in Table III.

(iii) Specification of identifiers as intrinsic terminal symbols to the compiler. The terminal phrase class < *I > causes a routine to attempt the identification of an identifier, instead of the letter by letter parsing.

A set of terminal phrase classes is defined which cause semantic action routines to be invoked at various points during execution time to speed the compilation process. < #15 > for example will invoke semantic action routine #15 whenever this action need to be taken at compile time.

This introduction is enough to interpret the IBNF system of the EIGENVALUE statement as given in Appendix A. For further details on TACOS, the reader may see reference [2].

Table III. IBNF Repetition Symbols

Repetition Character	Number of Occurrences	IBNF Example	BNF Interpretation
+	≥ 1	<A> ::= <B>+	<A> ::= <B>|<A> <A> <B>
*	≥ 0	<A> ::= <B>*	<A> ::= <empty>|<A> <B>
?	0 or 1	<A> ::= <B>?	<A> ::= <empty>|<B>

4.3 Compilation of the Eigenvalue Statement

From now on the reader may reference Appendices A and B that show the syntax and associated semantics for the eigenvalue statement. It will also be helpful to reference the examples shown in Appendix D.

We will discuss the implementation of an eigenvalue statement, with the implementation of the others being accomplished in a similar fashion,

Consider the following OL/2 statement:

EIGENVALUES α*A*B SMALLEST 3*γ → W;

After the recognition of the keywords EIGENVALUES, some flags and counters are set. The parser continues scanning looking for identifiers or array expressions. For identifiers, syntactic and semantic analyis is performed.

The array expression process is divided into three phases. First, the syntactic and semantic analysis of the expression, which transforms the source code into an expression tree. Second, this tree is modified in order to allow an optimized coding by the third pass. Last, the coding routine transforms this optimized tree code representation into a sequential list of three operand code (operand, operand, result). Reference [3] is a detailed treatment of this subject. The information extracted then consists of the pointer to the ACB of either the identifier or what will be the result at execution time of the array expression together with the following:

1) A negate flag which indicates whether or not the result of the expression has to be negated.
2) A transpose flag which indicates whether or not the result of the expression has to be transposed.
3) A storage flag which indicates if the space actually occupied by the result of the expression is temporary or not. Note that for array expressions this bit is always set to false in contrast with the bit associated to identifiers, which is always set to true.
4) A modulus number which indicates whether the expression or identifier is a member of a sequence of arrays.

After having recognized these elements successfully, the parser will continue scanning looking for either "→" or the keywords "SMALLEST", "LARGEST" or "IN". After the recognition of the keyword SMALLEST, some flags are set and a process similar to the one described above for OL/2 arithmetic expressions will handle the OL/2 scalar expression. After the recognition of the "→", the parser will look for any OL/2 identifiers that can appear on the right side (up to three), check for syntax and semantic errors, and extract information similar to that extracted for the OL/2 array expression, i.e., a set of pointers to ACB, negate, transpose, storage flags, and modulus number. The code generated is described in the next section.

It is clear that because of the fact that arbitrary array and scalar expressions or identifiers are imbedded in the statement, very little error checking can be done at compile time, so through the different ACB pointers, the rest of the error checking is done at run time. Appendix C contains the run time routines which accomplished this.

4.4 Code Generation

The code generated for the EIGENVALUE statement is in all cases a call to the @EIGEN routine. As was explained before, the information passed to this routine is summarized below.

i) ACB information for the array expressions or array or subarray references, in the form of pointers. These are of the form $XYZ or $TEMPXY. The former stands for a declared ACB pointer variable for the OL/2 array identifier Y declared by the programmer in block number Z and block

level X. The latter stands for a temporary ACB pointer variable for the Yth temporary variable whose array dimension is X.

Together with this go the negate, transpose, storage and modulus flags.

ii) Specific case number (one to five).

iii) For case three a flag indicating either SMALLEST or LARGEST and the scalar associated with the request.

iv) For cases four and five interval bounds.

Once generated, and referring to the program given in section 4.1, the PL/I call statements look as follows:

```
/ *  EIGENSYSTEM L → U, W;  * /

      CALL @EIGEN ('1'B, '0'B, '0'B, 0, $1U, 2, '0'B, 0, 0.0, 0.0,
                   '1'B, '0'B, '0'B, 0, $1U1, '1'B, '0'B, 0.B, 0,
                   $1W1, '0'B, '0'B, '0'B, 0, $0LNULL);

/ * EIGENVALUES  A<2,1> * A<1,2> → U<2>, V<2>, W<2,2>; * /
      CALL @0L0M0 ('1'B, '0'B, '0'B, 0, @0L2_LOCATE-SUBARRAY($1A1, 2, 1),
                   '1'B, '0'B, '0'B, 0, @0L2_LOCATE_SUBARRAY($1A1, 1, 2),
                   '1'B, '0'B, '0'B, 0, $TEMP2_1);

      CALL @EIGEN ('0'B, '0'B, '0'B, 0, $TEMP2_1, 1, '0'B, 0, 0.0,
                   0.0, '1'B, '0'B, ;0;B, 0, @0L2_LOCATE-SUBARRAY $1U1,2,0
                   '1'B, '0'B, '0'B, 0, @0L2_LOCATE_SUBARRAY($1V1, 2,0),
                   '1'B, '0'B, '0'B, 0, @0L2_LOCATE-SUBARRAY($1W1, 2, 2);
```

```
/ * EIGENVALUES  L  SMALLEST 2*S → Z; * /
      CALL @EIGEN ('1'B, '0'B, '0'B, 0, $ILI, 3, '0'B, (2*S),
                   0.0, 0.0, '1'B, '0'B, '0'B, 0, $LZL, '0'B, '0'B, '0'B,
                   0, $OLNULL, '0'B, '0'B, '0'B, 0, $OLNULL);
```

At run time the rest of the error checking is performed by this routine through the ACB's pointers. This error checking mainly consists of detecting any discrepancies between the shape of the source array, the associated request (case number, the associated scalars, interval bounds), and the characteristics of the target arrays.

Once free of errors, we perform a copy operation, which is fast and efficient, and pass control to the computational routines. Once the computation is finished hopefully without interruption, a copy operation must be performed from the elements containing the result into the target arrays.

The results so obtained can be used in other parts of the OL/2 program and can also be referenced or operated on in arbitrary manner. This completes the processing of the EIGENVALUE statement.

DISCUSSION

The different eigenvalue problems can be solved with the very-high level language facilities provided by OL/2. The user can combine the partitioning and easy formulation of array expressions capabilities of the language with these statements. This has been accomplished for the mode (real) now available for the different OL/2 types. Once the

complex mode is implemented, new additions can be made to enhance the power and generality.

The idea has been to take advantage of the existing subroutine packages (EISPACK) to provide by means of the specific language OL/2, a framework for the easy and elegant solution of the eigenvalue problem. It should be clear that the same concept can be applied to other collections of subroutines with great benefit to users who are primarily interested in using well defined, high quality, mathematical software.

REFERENCES

[1] H. C. Adams II, Dynamic Partitioning in the Array Language OL/2. M.S. Thesis, University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 421, January 1971.

[2] J. L. Gaffney Jr., "TACOS: A Table Driven Compiler-Compiler System". M.S. Thesis, University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 352, June 1969.

[3] D. R. Jurich, An Approach to the Compilation of Array Expressions in the OL/2 Language. M.S. Thesis, University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 550, September 1972.

[4] J. R. Phillips, The Structure and Design Philosophy of OL/2 - An Array Language - Part I: Language Overview. University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 544, October 1972.

[5] J. R. Phillips, The Structure and Design Philosophy of OL/2 - An Array Language - Part II: Algorithms for Dynamic Partitioning. University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 420, September 1971.

[6] J. R. Phillips, OL/2 User Guide. University of Illinois at Urbana-Champaign, Department of Computer Science (to appear).

[7] J. R. Phillips and H. C. Adams, Dynamic Partitioning for Array Languages, CACM, December 1972.

[8] B. T. Smith , J. M. Boyle, B. S. Garbow, Y. Ikebe, V. Klema, and C. Moler, Matrix Eigensystem Routines - EISPACK Guide. Springer-Verlag, 1974.

[9] J. H. Wilkinson and C. Reinsch, Linear Algebra, Springer-Verlag, 1971.

APPENDIX A

OL/2 SYNTAX FOR THE EIGENVALUE STATEMENT

(FOR THE COMPLETE SYNTAX OF OL/2 SEE REFERENCES 1 AND 4)

```
<EIGEN_STMT> ::= ((.EIGENVALUES. <#254> <#250> | .EIGENSYSTEM. <#254>)
                  <CODED_OL2_ARITH_EXP> <#251>  ( '->' | <CASE_THREE> |
                  <CASE_FOUR_OR_FIVE> ) '->' ) <#252>  ( ',' <#259> |
                  <EXPRESSION_LIST> <CODED_OL2_MOD_IDENT> <#260> )+ )
                  <#253> ';'  ;

<CODED_OL2_ARITH_EXP> ::= <#163> <OL2_ARITHMETIC_EXPRESSION> <#243> ;

<CASE_THREE> ::= (.SMALLEST. <#255> | .LARGEST.) <#256> <EXPRESSION_LIST>
                 <CODED_OL2_SCALAR_EXP> <#257>  ;

<CASE_FOUR_OR_FIVE> ::= .IN. <#258> '(' ( ',' <#259> | <EXPRESSION_LIST>
                        <CODED_OL2_SCALAR_EXP> <#260> )+ ')'  ;

<EXPRESSION_LIST> ::= <#244> ;

<CODED_OL2_MOD_IDENT> ::= <#163> <MODIFIED_OL2_IDENTIFIER> <#246> ;

<CODED_OL2_SCALAR_EXP> ::= <#163> <OL2_ARITHMETIC_EXPRESSION> <#246> ;
```

APPENDIX B

SEMANTIC ACTION ROUTINES

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*  SEMANTIC ACTION ROUTINES 250 TO 260 USED FOR THE             */
/*  EIGENVALUE STATEMENT. THERE ARE FIVE POSSIBLE CASES.         */
/*  <EXP>  DENOTES ANY OL2 ARITHMETIC EXPRESSION OR              */
/*  IDENTIFIER.                                                  */
/*                                                               */
/*                                                               */
/*  1)  EIGENVALUES  <EXP>                                       */
/*  2)  EIGENSYSTEM  <EXP>                                       */
/*  3)  EIGENVALUES <EXP> LARGEST | SMALLEST    K                */
/*  4)  EIGENVALUES <EXP> IN (A,B)                               */
/*  5)  EIGENSYSTEM <EXP> IN (A,B)                               */
/*                                                               */
/*                                                               */
/*  ALL OF THE STATEMENTS HAVE AS TARGET ARRAYS OL/2             */
/*  IDENTIFIERS. THE NUMBER AND TYPE OF THESE IDENTIFIERS        */
/*  DEPEND ON WHICH TYPE OF EXPRESSION ONE IS OPERATING ON       */
/*                                                               */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*                                                               */
/*  SEMANTIC ACTION ROUTINE 250 IS ENTERED AFTER THE             */
/*  RECOGNITION OF THE KEYWORD EIGENVALUES AND SETS A            */
/*  FLAG TO BE USED BY OTHER ROUTINES                            */
/*                                                               */
/*                                                               */
/*****************************************************************/
```

```
ACTION_250:       EV=YES;
                  GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 251 IS ENTERED WHEN <EXP> HAS    */
        /*  BEEN SUCCESSFULLY PARSED. THE CODE GENERATED INCLUDES    */
        /*  TRANSPOSE,NEGATE AND TEMPORARY STORAGE FLAGS AND         */
        /*  MODULUS NUMBER AS WELL AS A POINTER TO THE ATTACHED      */
        /*  ACB FOR <EXP>                                            */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_251:       OUTPUT_BUFFER='CALL @EIGEN(' || TEMPSTRING;
                  NAMES_USED(-25)=YES;
                  GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 252 IS ENTERED WHEN RIGHT HAND   */
        /*  SIDE OF EIGENVALUE STATEMENT IS GOING TO BE PROCESSED.   */
        /*  CODE GENERATED CONVEYS INFORMATION ABOUT THE CASE        */
        /*  RECOGNIZED AND ASSOCIATED PARAMETERS.                    */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_252:       IF ¬SYMT THEN DO;
                    IF EV THEN REQUEST=',1';
                    ELSE REQUEST=',2';
                    REQUEST=REQUEST || ','''0''B,0,0.0,0.0';
                  END;
                  CTR,CTR1=0;   LMT=3;
                  OUTPUT_BUFFER=OUTPUT_BUFFER || REQUEST;
                  GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 253 IS ENTERED AFTER THE         */
        /*  RECOGNITION OF THE STATEMENT TERMINATOR. IT COMPLETES    */
        /*  THE CODE GENERATION.                                     */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_253:       DO I=1 TO 3-CTR;
                    OUTPUT_BUFFER=OUTPUT_BUFFER ||
                      ','''0''B,''0''B,''0''B, ,$OLNULL';
                  END;
                  OUTPUT_BUFFER=OUTPUT_BUFFER || ');' ;
                  CALL SKIP_AND_OUTPUT;
                  GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 254 IS ENTERED AFTER THE         */
        /*  RECOGNITION OF AN EIGENVALUE STATEMENT. SOME FLAGS       */
        /*  AND COUNTERS ARE RESET                                   */
        /*                                                           */
        /*                                                           */
        /*************************************************************/
```

```
ACTION_254:     EV,SZE,SYMT='0'B;  REQUEST='';
                CTR,CTR1,LMT=0;
                GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 255 IS ENTERED AFTER THE         */
        /*  RECOGNITION OF THE KEYWORD SMALLEST. A FLAG IS SET       */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_255:     SZE=YES;
                GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 256 IS ENTERED AFTER THE         */
        /*  RECOGNITION OF CASE THREE. CODE GENERATED INCLUDES       */
        /*  SMALLEST-LARGEST FLAG AND CASE NUMBER                    */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_256:     IF ¬EV THEN CALL #ERROR(69);
                IF SZE THEN REQUEST=',''0''B';
                ELSE REQUEST=',''1''B';
                REQUEST=',3' || REQUEST;
                SYMT=YES;
                GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 257 IS ENTERED AFTER             */
        /*  THE PARSING OF THE SCALAR EXPRESSION ASSOCIATED WITH     */
        /*  CASE 3. CODE GENERATED CONVEYS THAT INFORMATION.         */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_257:     REQUEST=REQUEST || ',' || TEMPSTRING || ',0.0,0.0';
                GO TO RETURN_TO_PARSER;
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINE 258 IS ENTERED WHEN A CASE=4 OR  */
        /*  CASE=5 IS RECOGNIZED. CODE GENERATED CONVEYS THAT        */
        /*  INFORMATION. SOME FLAGS ARE SET.                         */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_258:     SYMT=YES;    LMT=2;
                IF EV THEN REQUEST=',4,''0''B,0';
                ELSE REQUEST=',5,''0''B,0';
                GO TO RETURN_TO_PARSER;
```

```
        /*************************************************************/
        /*                                                           */
        /*                                                           */
        /*  SEMANTIC ACTION ROUTINES 259 AND 260 HANDLE RIGHT HAND   */
        /*  SIDE OF EIGENVALUE STATEMENT. THE CODE GENERATED         */
        /*  CONVEYS INFORMATION ABOUT THE TARGET ARRAYS.             */
        /*                                                           */
        /*                                                           */
        /*************************************************************/

ACTION_259:     CTR1=CTR1+1;
                IF CTR¬=CTR1 THEN CALL #ERROR(70);
                GO TO RETURN TO PARSER;
ACTION_260:     CTR=CTR+1;
                IF CTR > LMT THEN CALL #ERROR(71);
                REQUEST=REQUEST || ',' || TEMPSTRING;
                GO TO RETURN_TO_PARSER;
```

APPENDIX C

RUN TIME ROUTINES FOR EIGENVALUE STATEMENT

```
/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*    SUBROUTINE @EIGEN DOES THE ERROR CHECKING AT RUN TIME          */
/*    THE RESTRICTIONS THAT APPLY ARE:                               */
/*    THE SOURCE ARRAY MUST BE ONE OF THE TYPES SYMMETRIC, SQUARE    */
/*    LOWER TRIANGULAR, UPPER TRIANGULAR OR TRIDIAGONAL              */
/*    IF THE ORDER OF THE SOURCE MATRIX IS N THEN :                  */
/*    FOR CASE 1 THE ORDER OF THE TARGET VECTORS MUST BE N           */
/*    FOR CASE 2 THE ORDER OF THE TARGET VECTORS MUST BE N, AND      */
/*    THE TARGET MATRIX MUST BE SQUARE OF ORDER N                    */
/*    FOR CASE 3 THE ORDER OF THE TARGET VECTOR MUST BE EQUAL TOTHE  */
/*    NUMBER OF EXTREME EIGENVALUES TO BE COMPUTED                   */
/*    FOR CASE 4 THE ORDER OF THE TARGET VECTOR MUST BE N            */
/*    FOR CASE 5 THE ORDER OF THE TARGET VECTOR MUST BE N, AND THE   */
/*    TARGET MATRIX MUST BE SQURE OF ORDER N                         */
/*                                                                   */
/*    BESIDES THIS ERROR CHECKING, SUBROUTINE @EIGEN ALLOCATES       */
/*    STORAGE FOR THE ARRAYS ON WHICH EISPAC ROUTINES OPERATE ON     */
/*    AND CALLS THEM TOGETHER WITH THE COPY ROUTINES                 */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

@EIGEN:  PROCEDURE(SVO,NGO,TRO,MODO,$PTR1,REQUEST,FLAG,NUM,
          LB,UB,SV1,NG1,TR1,MOD1,$PTR2,SV2,NG2,TR2,MOD2,
          $PTR3,SV3,NG3,TR3,MOD3,$PTR4);


DCL ($PTR1,$PTR2,$PTR3,$PTR4) POINTER,#PTR POINTER STATIC,
    (NUM,REQUEST,MODO,#DIM,MOD1,MOD2,MOD3) FIXED BIN(15,0),
    (LB,UB) FLOAT BIN(53),
    (FLAG,SVO,NGO,TRO,SV1,NG1,TR1,SV2,NG2,TR2,SV3,NG3,TR3)
    BIT(1),
    1 #ROOT_NODE BASED( #PTR) ALIGNED,
     2 #NAME CHAR(24),
     2 #ATTRIBUTES,
      (3 #DEFINED,
       3 #STORAGE,
       3 #TEMP_VARIABLE,
       3 #BASE,
       3 #SCALE,
       3 #MODE) BIT(1),
       3 #PRECISION FIXED BIN(15,0),
     2 #LENGTH FIXED BIN(31,0),
    (2 #DIMENSIONALITY,
     2 #MODULUS,
     2 #TYPE,
     2 #ROW_INCR,
     2 #DIAG_INCR) FIXED BIN(15,0),
    (2 $ORIGIN, 2 $PARTITION_PTR) POINTER,
     2 #BOUND_PAIR  ( #DIM REFER ( #DIMENSIONALITY )),
      (3 #EXTENT,
       3 #LOWER,
       3 #UPPER) FIXED BIN(15,0);
 DCL HUH1 ENTRY(FLOAT BIN(53),FLOAT BIN(53),FIXED BIN(15),
                FIXED BIN(15));
 DCL TPN ENTRY(,FIXED BIN(15,0));
```

```
/* DECLARATIONS FOR THE OPERATIONAL EISPAC ROUTINES */

DCL BISECT ENTRY(FIXED BIN(15,0),FLOAT BIN(53),,,,FLOAT
    BIN(53),FLOAT BIN(53),FIXED BIN(15,0),FIXED BIN(15,0),
    ,,FIXED BIN(15,0),,),
    HQR ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FIXED
        BIN(15,0),FIXED BIN(15,0),,,,FIXED BIN(15,0)),
    HQR2 ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FIXED
         BIN(15,0),FIXED BIN(15,0),,,,,FIXED BIN(15,0)),
    IMTQL1 ENTRY(FIXED BIN(15,0),,,FIXED BIN(15,0)),
    IMTQL2 ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),,,,
           FIXED BIN(15,0)),
    ORTBAK ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FIXED
           BIN(15,0),,,FIXED BIN(15,0),),
    ORTHES ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FIXED
           BIN(15,0),FIXED BIN(15,0),,),
    ORTRAN ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FIXED
           BIN(15,0),FIXED BIN(15,0),,,),
    RATQR ENTRY(FIXED BIN(15,0),FLOAT BIN(53),,,,FIXED
           BIN(15,0),,,,BIT(1),FIXED BIN(15,0),FIXED
           BIN(15,0)),
    TRBAK1 ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),,,FIXED
           BIN(15,0),,),
    TRED1 ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),,,,),
    TRED2 ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),,,,),
    TSTURM ENTRY(FIXED BIN(15,0),FIXED BIN(15,0),FLOAT
           BIN(53),,,,FLOAT BIN(53),FLOAT BIN(53),FIXED
           BIN(15,0),FIXED BIN(15,0),,,FIXED BIN(15,0),,,,
           ,,);

DCL SPRSPEC ENTRY(PTR,FIXED BIN(31)) RETURNS(PTR);
DCL COPY ENTRY(BIT(1),BIT(1),FIXED BIN(15,0),PTR,
       PTR,FIXED BIN(15,0),FIXED BIN(15,0));
DCL @OL2ICS ENTRY(FIXED BIN(15),FIXED BIN(15));
DCL TESTSYM ENTRY(PTR) RETURNS(BIT(1));
DCL (PASS,SHAPE,PAR1 INIT(1),PAR2 INIT(1),PAR3 INIT(1),
     PAR4 INIT(1),PAR5 INIT(1),PAR6 INIT(1),PAR7 INIT(1),
    EXT11,EXT12,EXT21,EXT22,EXT31,EXT32,EXT41,EXT42,
    ZERO INIT(0),ONE INIT(1),TWO INIT(2),THREE INIT(3),
    FOUR INIT(4),FIVE INIT(5),SIX INIT(6),SEVEN INIT(7),
    TEN INIT(10),ELEVEN INIT(11),TYPE3,TYPE4) FIXED
    BIN(15,0),SYM BIT(1);


  #PTR=$PTR1;
  EXT11= #PTR -> #EXTENT(1); EXT12= #PTR ->#EXTENT(2);
  #PTR=$PTR2;
  EXT21= #PTR -> #EXTENT(1); EXT22= #PTR ->#EXTENT(2);
  #PTR=$PTR3;
  EXT31= #PTR -> #EXTENT(1); EXT32= #PTR ->#EXTENT(2);
  TYPE3= #PTR -> #TYPE;    #PTR=$PTR4;
  EXT41= #PTR -> #EXTENT(1); EXT42= #PTR ->#EXTENT(2);
  TYPE4= #PTR -> #TYPE;    #PTR=$PTR1;
  SHAPE=$PTR1 -> #TYPE;
  IF SHAPE ¬= ZERO THEN SYM='1'B;
  ELSE SYM='0'B;
  IF REQUEST=ONE | REQUEST=THREE | REQUEST=FOUR THEN
    PASS=15;  ELSE PASS=16;
  IF EXT11 < ZERO | EXT12 < ZERO | EXT21 < ZERO |
     EXT22 < ZERO  THEN CALL @OL2ICS(24,PASS);
  IF EXT11 ¬= EXT12 THEN CALL @OL2ICS(24,PASS);
  IF SHAPE=THREE | SHAPE=FOUR | SHAPE=SIX THEN CALL
     @OL2ICS(24,PASS);
  IF $PTR1 -> #LENGTH <= ONE THEN CALL @OL2ICS(25,PASS);
  IF SHAPE=FIVE THEN SYM=TESTSYM($PTR1);
```

```
      /* EIGENVALUES */
    IF REQUEST=ONE THEN DO;
       IF EXT11 ¬= EXT21 | EXT22 ¬= ZERO THEN CALL @CL2ICS(26,PASS);
       IF SHAPE=ZERO | ¬SYM THEN DO;
         IF EXT31 < ZERO | EXT32 < ZERO THEN CALL @OL2ICS(27,PASS);
         IF EXT11 ¬= EXT31 | EXT32 ¬= ZERO THEN CALL @OL2ICS(26,PASS);
       END;
       GO TO SKIP_REST;
    END;
      /* EIGENSYSTEM */
    IF REQUEST=TWO THEN DO;
     IF EXT11 ¬= EXT21 | EXT22 ¬= ZERO THEN CALL @OL2ICS(26,PASS);
     IF EXT31 < ZERO | EXT32 < ZERO THEN CALL @OL2ICS(27,PASS);
     IF SHAPE=ZERO | ¬SYM THEN DO;
       IF EXT11 ¬= EXT31 | EXT32 ¬=ZERO THEN CALL @CL2ICS(26,PASS);
       IF EXT41 < ZERO | EXT42 < ZERO THEN CALL @OL2ICS(27,PASS);
       IF EXT11 ¬= EXT41 | EXT42 ¬= EXT11 |
         TYPE4 ¬= ZERO THEN CALL @OL2ICS(26,PASS);
     END;
     ELSE DO;
         IF EXT11 ¬= EXT31 | EXT32 ¬= EXT11 |
           TYPE3 ¬= ZERO THEN CALL @OL2ICS(26,PASS);
     END;
     GO TO SKIP_REST;
    END;
      /* EXTREME EIGENVALUES */
   IF REQUEST=THREE THEN DO;
     IF SHAPE=ZERO | ¬SYM THEN CALL @CL2ICS(24,PASS);
     IF NUM <= ZERO | NUM > EXT11 THEN CALL @OL2ICS(28,PASS);
     IF EXT21¬=(NUM-ONE) | EXT22¬=ZERO THEN CALL @OL2ICS(26,PASS);
     GO TO SKIP_REST;
   END;
    /* EIGENVALUES OR EIGENSYSTEM IN AN INTERVAL */
     IF SHAPE=ZERO | ¬SYM THEN CALL @OL2ICS(24,PASS);
     IF (UB-LB)<=0.0E0 THEN CALL @OL2ICS(29,PASS);
     IF EXT22 ¬= ZERO THEN CALL @OL2ICS(26,PASS);
     IF REQUEST=FIVE THEN DO;
       IF EXT31 < ZERO | EXT32 < ZERO THEN CALL @OL2ICS(27,PASS);
       IF EXT11 ¬= EXT31 | TYPE3 ¬= ZERO
         THEN CALL @OL2ICS(26,PASS);
     END;
SKIP_REST: /* HERE TO ALLOCATE STORAGE, COPY AND EXECUTE */
  EXT11=EXT11+ONE;
  IF SHAPE=ZERO | ¬SYM THEN DO;
    BEGIN;
          DCL (A(EXT11,EXT11),ORT(EXT11),WR(EXT11),Z(PAR1,PAR1),
              WI(EXT11)) FLOAT BIN(53),IERR FIXED BIN(15,0);
          IERR=ZERO;
          CALL COPY(SVO,TRO,MODO,$PTR1,ADDR(A),SHAPE,ZERO);
          IF SHAPE=ZERO & TRO THEN CALL TPN(A,EXT11);
          CALL ORTHES(EXT11,EXT11,ONE,EXT11,A,ORT);
          IF REQUEST=ONE THEN DO;
            CALL HQR(EXT11,EXT11,ONE,EXT11,A,WR,WI,IERR);
            IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
          END;
          ELSE DO;
                  CALL ORTRAN(EXT11,EXT11,ONE,EXT11,A,ORT,Z);
                  CALL HQR2(EXT11,EXT11,ONE,EXT11,A,WR,WI,Z,IERR);
                  IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
                  CALL ORTBAK(EXT11,ONE,EXT11,A,ORT,EXT11,Z);
                  CALL COPY(SV3,TR3,MOD3,$PTR4,ADDR(Z),
                            TEN,EXT11-ONE);
          END;
```

```
    IF NGO THEN DO;
       WR=-WR;    WI=-WI;
    END;
    CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(WR),TEN,ZERO);
    CALL COPY(SV2,TR2,MOD2,$PTR3,ADDR(WI),TEN,ZERO);
  END;
  RETURN;
END;

   /* SYMMETRIC MATRICES */
  BEGIN;
       DECLARE (A(PAR1,PAR1),D(EXT11),E(EXT11),E2(EXT11),
              Z(PAR2,PAR2),W(PAR3),BD(PAR4),RV1(PAR5),
              RV2(PAR5),RV3(PAR5),RV4(PAR6),RV5(PAR6),
              RV6(PAR5))
              FLOAT BIN(53),IND(PAR7) FIXED BIN(15),
              (IERR,APRX) FIXED BIN(15,0);
        IF SHAPE=FIVE THEN DO;
          CALL COPY('1'B,TRO,MODO,$PTR1,ADDR(D),SIX,ZERO);
          CALL COPY(SVO,TRO,MODO,$PTR1,ADDR(E),ELEVEN,ZERO);
          DO I=TWO TO EXT11-ONE;
             E2(I)=E(I)*E(I);
          END;
        END;
        ELSE DO;
          CALL COPY(SVO,TRO,MODO,$PTR1,ADDR(A),SHAPE,ZERO);
          IF REQUEST ¬=TWO THEN CALL TRED1(EXT11,
            EXT11,A,D,E,E2);
          ELSE CALL TRED2(EXT11,EXT11,A,D,E,Z);
        END;
        IERR=ZERO;
        IF REQUEST=ONE THEN DO;
          CALL IMTQL1(EXT11,D,E,IERR);
          IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
          IF NGO THEN D=-D;
          CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(D),TEN,ZERO);
          RETURN;
        END;
        IF REQUEST=TWO THEN DO;
          IF SHAPE=FIVE THEN DO;
            DO I=1 TO EXT11;
              Z(I,I)=1.0E0;
            END;
          END;
          CALL IMTQL2(EXT11,EXT11,D,E,Z,IERR);
          IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
          IF SHAPE¬=FIVE THEN CALL TRBAK1(EXT11,
            EXT11,A,E,EXT11,Z);
          IF NGO THEN D=-D;
          CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(D),TEN,ZERO);
          CALL COPY(SV2,TR2,MOD2,$PTR3,ADDR(Z),TEN,EXT11-1);
          RETURN;
        END;
        IF REQUEST=THREE THEN DO;
            CALL RATQR(EXT11,1.0E-07,D,E,E2,NUM,W,
                       IND,BD,FLAG,ZERO,IERR);
            IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
            IF NGO THEN W=-W;
            CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(W),TEN,ZERO);
          RETURN;
        END;
```

```
            IF REQUEST=FOUR THEN DO;
                APRX=ZERO;
                CALL BISECT(EXT11, 1.0E-07,D,E,E2,LB,UB,
                            EXT11,APRX,W,IND,IERR,RV4,RV5);
                IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
                IF APRX ¬= (EXT21+1) THEN CALL HUH1(LB,UB,
                        APRX,EXT11);
                IF NGO THEN W=-W;
                CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(W),TEN,APRX-1);
              RETURN;
            END;
            DO;
                CALL TSTURM(EXT11,EXT11,1.0E-07,D,E,E2,
                    LB,UB,EXT11,APRX,W,Z,IERR,RV1,RV2,
                     RV3,RV4,RV5,RV6);
                IF IERR¬=ZERO THEN CALL @OL2ICS(30,PASS);
                IF NGO THEN W=-W;
                IF SHAPE¬=FIVE THEN CALL TRBAK1(EXT11,
                  EXT11,A,E,APRX,Z);
                IF APRX ¬= (EXT21+1) THEN CALL HUH1(LB,
                   UB,APRX,EXT11);
                CALL COPY(SV1,TR1,MOD1,$PTR2,ADDR(W),TEN,APRX-1);
                CALL COPY(SV2,TR2,MOD2,$PTR3,ADDR(Z),TEN,APRX-1);
              RETURN;
            END;
          END;
END @EIGEN;
```

APPENDIX D

EXAMPLE OF EIGENVALUE STATEMENTS

```
***OL/2 OPERATOR LANGUAGE/2  UNIVERSITY OF ILLINOIS  VER. 3.1

EXAMPLE:   MAIN PROCEDURE;

          LET B AND C BE MATRICES OF ORDER(6),

            AND X AND Y BE VECTORS OF ORDER(6),

            AND W BE A VECTOR OF ORDER(2);

          INPUT B;

              EIGENVALUES  B*B  ->  X , Y ;

                          OUTPUT  X , Y  .  :  PACK;

              SET L TO THE LOWER TRIANGULAR PART OF B;

              EIGENSYSTEM  L  ->  X , C ;

              SET U TO THE UPPER TRIANGULAR PART OF B;

              EIGENVALUES  U  SMALLEST  2  ->  W ;

                          OUTPUT  X , C , W  :  PACK;

    END EXAMPLE;
OL/2 DIAGNOSTICS.

NO ERRORS DETECTED.

END OF DIAGNOSTICS.
CORE NOT USED BY COMPILER WAS:
 48K BYTES.


 END OF COMPILATION.
 COMPILE TIME:   2.78  SEC.
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-703	2.	3. Recipient's Accession No.

4. Title and Subtitle

THE EIGENVALUE PROBLEM IN THE OL/2 LANGUAGE

5. Report Date: May 1975

6.

7. Author(s)

Ricardo Macias Carrasco

8. Performing Organization Rept. No. UIUCDCS-R-75-703

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

US NSF GJ-328

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered

Master's Thesis

14.

15. Supplementary Notes

16. Abstracts

This report is concerned with the eigenvalue problem in the context of the very high level, array language OL/2. Using the capabilities that are within the OL/2 language together with the high quality software that is present in EISPACK, an implementation of the eigenvalue problem is presented. The syntax and semantics of the various eigenvalue statements is given along with the parsing of these statements, the generated code, the data structures, and the particular algorithms that are used.

17. Key Words and Document Analysis. 17a. Descriptors

Eigenvalue Software
Array Language
Very High Level Language
Eigenvalue-Eigenvector Syntax

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	40
	20. Security Class (This Page) UNCLASSIFIED	22. Price